# The Promises and Pitfalls of Deep Kernel Learning: Supplementary Material

**Sebastian W. Ober**[1] **Carl E. Rasmussen**[1,2] **Mark van der Wilk**[3]

[1]Department of Engineering, University of Cambridge, Cambridge, United Kingdom
[2]Secondmind.ai, Cambridge, United Kingdom
[3]Department of Computing, Imperial College London, London, United Kingdom

## A PROOF OF PROPOSITION 1

We restate the proposition:

**Proposition 1.** *Consider the GP regression model as described in Eq. 1. Then, for any valid kernel function that can be written in the form $k(x, x') = \sigma_f^2 \hat{k}(x, x')$, where $\sigma_f^2$ is a learnable hyperparameter along with learnable noise $\sigma_n^2$ (and any other kernel hyperparameters), we have that the "data fit" term will equal $-N/2$ (where $N$ is the number of datapoints) at the optimum of the marginal likelihood.*

*Proof.* We reparameterize $\sigma_n^2 = \hat{\sigma}_n^2 \sigma_f^2$. Then, writing $K + \sigma_n^2 I_N = \sigma_f^2(\hat{K} + \hat{\sigma}_n^2 I_N)$, the result follows by differentiating the log marginal likelihood with respect to $\sigma_f^2$:

$$\begin{aligned}\frac{d}{d\sigma_f^2}\log p(\mathbf{y}) &= \frac{d}{d\sigma_f^2}\left(-\frac{N}{2}\log\sigma_f^2 - \frac{1}{2}\log|\hat{K} + \hat{\sigma}_n^2 I_N| - \frac{1}{2\sigma_f^2}\mathbf{y}^T(\hat{K} + \hat{\sigma}_n^2 I_N)^{-1}\mathbf{y}\right) \\ &= -\frac{N}{2\sigma_f^2} + \frac{1}{2\sigma_f^4}\mathbf{y}^T(\hat{K} + \hat{\sigma}_n^2 I_N)^{-1}\mathbf{y}.\end{aligned}$$

Setting the derivative equal to zero gives:

$$\sigma_f^2 = \frac{1}{N}\mathbf{y}^T(\hat{K} + \hat{\sigma}_n^2 I_N)^{-1}\mathbf{y}.$$

Substituting this into the data fit term gives the desired result. □

We note that this result was essentially proven in Moore et al. [2016], although they did not consider the last step of substituting the result into the data fit term. Instead, they used the result as a means of analytically solving for the optimal signal variance to reduce the number of parameters and hence speed up optimization.

## B EXPERIMENTAL DETAILS

All experiments on real datasets (UCI, CIFAR-10, UTKFace) were written in TensorFlow 2 [Abadi et al., 2015], using GPflow [Matthews et al., 2017] to implement the DKL models. We use jug [Coelho, 2017] to easily run the experiments. The experiments were run on single GPUs using both NVIDIA Tesla P100-PCIE-16GB GPUs and NVIDIA GeForce RTX 2080 Ti GPUs.

### B.1 DATASETS

We describe the datasets used as well as the splits and preprocessing.

**Toy dataset** The toy dataset is that as introduced in Snelson and Ghahramani [2006], which was generated from a function realization sampled from a SE GP. The dataset comprises 200 input-output pairs and can be found at `http://www.gatsby.ucl.ac.uk/~snelson/`. We normalize both inputs and outputs for training and plot the unnormalized values and predictions.

**UCI** We use a slightly modified version of Bayesian Benchmarks (`https://github.com/hughsalimbeni/bayesian_benchmarks`) to obtain the UCI datasets we use. The modification is to rectify minor data leakage in the normalization code: they normalize using the statistics from the entire dataset before dividing into train/test splits, instead of normalizing using only the train split statistics. We perform cross-validation using 20 90%/10% train/test splits, and report means and standard errors for each metric. Note that we report metrics on the normalized datasets to lead to more interpretable results: namely, an RMSE of 1 corresponds to predicting 0 for each test point.

**CIFAR-10** We use the standard CIFAR-10 dataset [Krizhevsky and Hinton, 2009], with the standard train-validation split of 50,000 and 10,000 images, respectively, using the validation split as the test set, as is common practice. We preprocess the images by simply dividing the pixel values by 255, so that each value lies between 0 and 1.

**UTKFace** The UTKFace dataset [Zhang et al., 2017] is a large face dataset consisting of 23,708 images of faces, annotated with age, gender, and ethnicity. The faces have an age range from 0 to 116. We use the aligned and cropped version to limit the amount of preprocessing necessary, available at `https://susanqq.github.io/UTKFace/`. These cropped images have sizes $200 \times 200 \times 3$. We choose 20,000 images to be in the train dataset, with the remaining being used for testing. We again perform preprocessing by dividing the pixel values by 255, and we additionally normalize the age values. The metrics we report all use the normalized values, as with the UCI datasets.

### B.2 MODELS

We describe the models used for the experiments. To ensure that the comparisons between neural networks and DKL models are as fair as possible, we ensure that each model used in direct comparison has the same number of layers: for the DKL models, we remove the last fully-connected layer of the neural network and replace it with the ARD SE GP. All neural networks use ReLU activations. For all SVDKL models, the inducing points live in the neural network feature space at the input to the GP.

**Toy dataset** We use an architecture of $[100, 50, 2]$ for the hidden-layer widths for the neural network. For DKL, we use the pre-activation features of the final hidden layer for the input to the GP.

**UCI** For BOSTON, ENERGY, we use a ReLU architecture of $[50, 50]$, and a ReLU architecture $[1000, 500, 50]$ for KIN40K, POWER, and PROTEIN. We note that these architectures are smaller than the ones proposed by Wilson et al. [2016] For the SVGP baseline, we use an ARD SE kernel, with 100 inducing points for the small datasets (BOSTON, ENERGY) and 1000 inducing points for the larger ones. For the DKL models, we use the post-activation features from the final hidden layer as inputs to the GPs, which use ARD SE kernels. For SVDKL, we initialize the inducing points using the k-means algorithm on a subset of the training set. We use 100 inducing points for the smaller datasets (BOSTON, ENERGY) and 1000 on the larger ones. The method for initializing the inducing points, and the number of inducing points, is the same for the SVGP baseline model, which uses a standard ARD SE kernel.

**CIFAR-10** We use a modified ResNet18 [He et al., 2016] architecture as the baseline neural network architecture; the main modification is that we have added another fully-connected layer at the output to ensure that the neural network and DKL models are comparable in depth. Therefore, instead of the standard single fully-connected layer after a global average pooling layer, we have two fully-connected layers. While we could take the output of the global average pooling layer, this is typically very high-dimensional and thus potentially unsuitable as an input to a GP. For most experiments, we fix the width of the last hidden layer (the final feature width) to 10, although we do consider changing that in App. C. We additionally add batchnorm layers [Ioffe and Szegedy, 2015] before the ReLU activations in the residual blocks. For SVDKL, we use 1000 inducing points initialized with k-means on a subset of the training set. As with UCI, the features at the input to the GP are post-activation features. For all classification models, we use softmax activation to obtain probabilities for the cross-entropy loss, and for the SVDKL models we use 10 samples from the latent function posterior to compute the log likelihood term of the ELBO.

**UTKFace** As with CIFAR-10, we again use a modified ResNet18 [He et al., 2016] architecture with an additional fully-connected layer at the output. For SVDKL, we again use 1000 inducing points.

### B.3 IMPLEMENTATION DETAILS

All models are optimized using ADAM [Kingma and Ba, 2015]. Throughout, we try to ensure that we train each model for comparable numbers of gradient steps and learning rates.

**Toy dataset** We train both the NN and DKL models in for 10,000 gradient steps using learning rates of 0.001. No weight decay was used. For the HMC experiments, we use a step size of 0.005, 20 leapfrog steps, and a prior variance of 1 on the network weights. We burn in for 10,000 samples, then use 1,000 iterations to sample, thinned by a factor of 10.

**UCI** For each model, we train with an initial learning rate of 0.001. For BOSTON and ENERGY, we use a batch size of 32 and train the minibatched algorithms for a total of 400 epochs, and use a learning rate scheduler that decreases the learning rate by a factor of 10 after 200 and 300 epochs. For KIN40K, POWER, and PROTEIN we use a batch size of 100, training for 160 epochs with the same learning rate schedule that triggers at 80 and 120 epochs. For the full-batch methods, we ensure that they are trained for the lesser of the same number of gradient steps or 8000 gradient steps (due to limited computational budget), with the learning rate schedule set to trigger at the corresponding number of gradient steps as the batched methods. This ensures a fair comparison when claiming that the full-batch methods overfit in comparison to the stochastic versions. For the deep models, we use a weight decay of 1e-4 on the neural network weights. We do not use any pretraining for the DKL models, as we did not find it necessary for these datasets. We initialize the log noise variance to -4 for the DKL models. We train the neural network models using mean squared error loss, and use the maximum likelihood noise estimate after training to compute train and test log likelihoods.

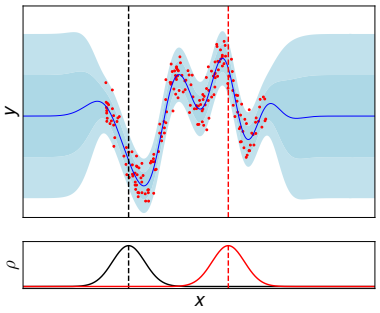
**CIFAR-10** We describe the details for batch size 100; for batch size 500, we ensure that we use the same number of gradient steps. We do note use weight decay as we found that it hurt test accuracy. For NN and SVDKL, we train for 160 epochs total: we decrease the learning rate from the initial 1e-3 by a factor of 10 at 80 and then 120 epochs. For pNN, we train for an additional 160 epochs in the same way (restarting the learning rate at 1e-3). For pSVDKL, we start by training with the neural network parameters fixed for 80 epochs, with learning rate decreases at 40 and 60 epochs. We then reset the learning rate to 1e-3, and train for an additional 80 epochs with learning rate decreases at 40 and 60 epochs. For the experiments with data augmentation, we use random horizontal flipping and randomly crop $32 \times 32 \times 3$ images from the original images padded up to $40 \times 40 \times 3$.

We use the same losses as the potentials for SGLD. We use the trained NNs to initialize the weights to reasonable values, and set the batch size to 100. We initialize the learning rate to 1e-3 (which we then scale down by the dataset size to account for the scale of the potential), and decay the learning rate at each epoch by a factor of $1/(1 + 0.4 \times \text{epoch})$ to satisfy Robbins-Monro. For the NN, we burn in for 100 epochs, and then sample every other epoch for 100 epochs, leading to 50 samples. For SVDKL, we follow the approach in Hensman et al. [2015], and learn the variational parameters (i.e. 1000 inducing points) and GP hyperparameters with the fixed, pretrained NN weights, using the same hyperparameters as for pSVDKL. We then follow the SGLD approach we took for the NN, with 100 epochs of burn in and 100 epochs of sampling, starting with a learning rate of 1e-3.

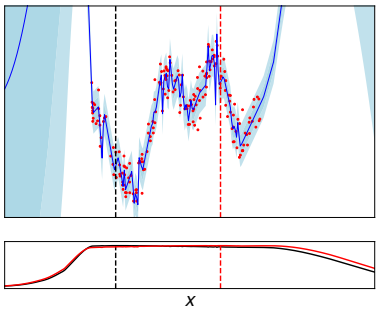
**UTKFace** We follow the same approach as for CIFAR-10. We list the minor differences. We use a small weight decay of 1e-4. For the SVDKL models, we initialize the log noise variance to -4. We use mean squared error loss for the NNs; however, for SGLD we use a Gaussian likelihood with log noise variance initialized to -4. For the SGLD experiments, we initialize the learning rate to 1e-5. For data augmentation, we again use random horizontal flipping as well as randomly cropping $200 \times 200 \times 3$ images from the original images padded up to $240 \times 240 \times 3$.

## C ADDITIONAL EXPERIMENTAL RESULTS

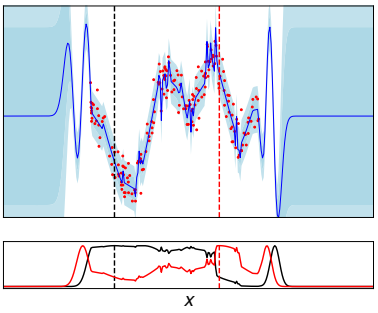
Here we briefly present some additional experimental results.



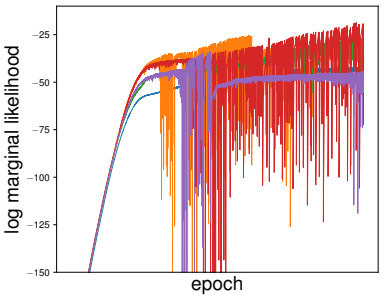

(a) SE kernel. LML $= -89.3$



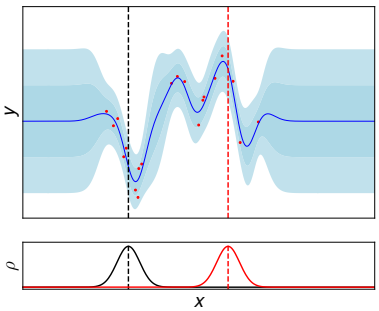

(b) Example DKL. LML $= -23.4$



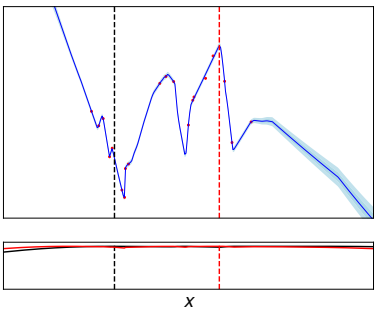

(c) Example DKL. LML $= -45.7$




(d) Training curves for 5 different initializations for DKL on the full toy dataset




(e) SE kernel. LML $= -17.5$




(f) Example DKL. LML $= 28.3$

Figure 1: Plots of fits and training curves using standard SE kernel and DKL. Below each fit we plot two correlation functions $\rho_{x'}(x) = k(x, x')/\sigma_f^2$ induced by each kernel, where the location of $x'$ is given by the dashed vertical lines. (a)-(d) show fits and training curves for the full toy dataset introduced in Snelson and Ghahramani [2006], whereas (e)-(f) show fits on the subsampled version from Titsias [2009].

Table 1: Results for UTKFace.

| | ELBO | Train RMSE | Test RMSE | Train LL | Test LL |
|---|---|---|---|---|---|
| NN-2 | - | 0.37 ± 0.26 | 0.61 ± 0.16 | 0.58 ± 0.82 | -23.14 ± 10.94 |
| SVDKL-2 | -0.22 ± 1.00 | 0.39 ± 0.25 | 0.61 ± 0.16 | 0.14 ± 0.68 | -3.60 ± 1.90 |
| pNN-2 | - | 0.36 ± 0.26 | 0.61 ± 0.16 | 0.74 ±0.88 | -35.66 ± 14.00 |
| fSVDKL-2 | -0.32 ± 4.00 | 0.45 ± 0.22 | 0.45 ±0.23 | -0.26 ± 0.50 | -0.23 ±0.52 |
| pSVDKL-2 | 0.36 ±3.00 | 0.36 ±0.26 | 0.53 ± 0.20 | 0.47 ± 0.77 | -3.24 ± 0.76 |
| NN-5 | - | 0.04 ±0.00 | 0.41 ± 0.01 | 1.72 ± 0.05 | -42.86 ± 2.79 |
| SVDKL-5 | -0.17 ± 0.59 | 0.11 ± 0.01 | 0.47 ± 0.01 | 0.40 ± 0.11 | -1.41 ± 0.16 |
| pNN-5 | - | 0.04 ±0.00 | 0.41 ± 0.00 | 1.79 ±0.01 | -48.92 ± 0.62 |
| fSVDKL-5 | 0.31 ± 0.48 | 0.17 ±0.00 | 0.17 ±0.00 | 0.38 ± 0.02 | 0.37 ±0.03 |
| pSVDKL-5 | 0.99 ±0.73 | 0.04 ±0.00 | 0.32 ± 0.01 | 1.18 ± 0.06 | -2.61 ± 0.26 |
| NN-10 | - | 0.04 ±0.00 | 0.40 ± 0.00 | 1.81 ± 0.01 | -48.73 ± 1.64 |
| SVDKL-10 | 0.92 ± 0.15 | 0.04 ±0.00 | 0.40 ± 0.01 | 1.30 ± 0.01 | -6.88 ± 0.38 |
| pNN-10 | - | 0.04 ±0.00 | 0.41 ± 0.00 | 1.83 ±0.01 | -53.72 ± 1.71 |
| fSVDKL-10 | 1.05 ±0.02 | 0.08 ± 0.03 | 0.31 ±0.07 | 1.16 ± 0.31 | -7.55 ± 3.42 |
| pSVDKL-10 | 1.03 ± 0.07 | 0.04 ±0.00 | 0.38 ± 0.02 | 1.20 ± 0.08 | -4.74 ±1.35 |
| NN-20 | - | 0.04 ±0.00 | 0.40 ± 0.00 | 1.78 ± 0.01 | -46.77 ± 1.72 |
| SVDKL-20 | 0.22 ± 0.01 | 0.08 ± 0.02 | 0.43 ± 0.01 | 0.78 ± 0.22 | -3.42 ±1.52 |
| pNN-20 | - | 0.04 ±0.00 | 0.41 ± 0.00 | 1.80 ±0.01 | -50.51 ± 0.47 |
| fSVDKL-20 | 0.71 ± 0.30 | 0.12 ± 0.03 | 0.24 ±0.06 | 0.83 ± 0.34 | -5.06 ± 4.46 |
| pSVDKL-20 | 1.15 ±0.10 | 0.04 ±0.00 | 0.34 ± 0.03 | 1.33 ± 0.04 | -4.16 ± 0.42 |
| NN-50 | - | 0.04 ±0.00 | 0.40 ± 0.00 | 1.79 ± 0.01 | -47.72 ± 0.68 |
| SVDKL-50 | 0.92 ± 0.27 | 0.04 ±0.00 | 0.40 ± 0.00 | 1.33 ± 0.03 | -7.35 ± 0.45 |
| pNN-50 | - | 0.04 ±0.00 | 0.41 ± 0.01 | 1.81 ±0.00 | -51.14 ± 1.33 |
| fSVDKL-50 | 1.14 ± 0.31 | 0.08 ± 0.03 | 0.32 ±0.06 | 1.29 ± 0.35 | -11.60 ± 4.91 |
| pSVDKL-50 | 1.21 ±0.03 | 0.04 ±0.00 | 0.37 ± 0.02 | 1.37 ± 0.02 | -5.71 ±0.55 |

### C.1 TOY

We show additional plots of fits and training curves for the toy problem from Snelson and Ghahramani [2006] in Fig. 1. Below the fits, we again show the kernel correlation at two different points $x'$, marked by the vertical dashed lines. In Figure 1a, we show the fit using the standard squared exponential kernel, followed by two fits using DKL in Figures 1b and 1c. In Fig. 1d we show training curves for 5 different initializations; note that unlike in the main text we use a learning rate of 1e-4 here and so require more training iterations to converge. Finally, we consider plots of fits on the subsampled version of the dataset as in Titsias [2009]; we show fits using the SE kernel and DKL in Figures 1e and 1f, respectively. For each fit, we also show the log marginal likelihood in the caption.

We make a few observations. First, note that different initializations can lead to very different fits and LMLs. Moreover, as predicted by our theory, the highest log marginal likelihoods are obtained when the prior attempts to correlate all the points in the input domain: Fig. 1b obtains a higher LML than Fig. 1c. However, instability in training often leads to worse LMLs than could be obtained (Fig. 1d). Finally, we note that the overfitting is substantially worse on the subsampled version of the dataset: we also see that the prior is more correlated than previously (Fig. 1f).

### C.2 CHANGING THE FEATURE DIMENSION

We perform experiments changing the feature dimension $Q$ for the UTKFace and CIFAR-10 datasets. We present the results in Tables 1 and 2, where each model name is followed by the feature space dimension. For UTKFace, it is clear that 2 neurons is not sufficient to fit the data. Beyond 2, we see only minor changes in performance. For CIFAR-10, we find that we need at least 10 neurons to fit well, but beyond 10 there are again only minor differences. We choose 10 neurons for both experiments out of convenience.

Table 2: Results for CIFAR-10.

| | ELBO | Train Acc. | Test Acc. | Train LL | Test LL | Inc. Test LL | ECE |
|---|---|---|---|---|---|---|---|
| NN-2 | - | 0.69 ± 0.24 | 0.51 ± 0.17 | -0.81 ± 0.61 | -6.42 ± 2.23 | -6.40 ± 1.78 | 0.14 ± 0.06 |
| SVDKL-2 | -1.38 ± 0.38 | 0.52 ± 0.17 | 0.46 ± 0.15 | -1.34 ± 0.39 | -1.57 ±0.30 | -2.61 ±0.13 | 0.04 ±0.02 |
| pNN-2 | - | 0.70 ±0.24 | 0.53 ±0.17 | -0.77 ±0.63 | -5.25 ± 1.36 | -7.28 ± 2.03 | 0.15 ± 0.06 |
| fSVDKL-2 | -0.85 ± 0.59 | 0.69 ± 0.24 | 0.51 ± 0.17 | -0.81 ± 0.61 | -1.83 ± 0.20 | -4.35 ± 0.84 | 0.14 ± 0.06 |
| pSVDKL-2 | -0.78 ± 0.62 | 0.70 ±0.24 | 0.53 ±0.17 | -0.78 ± 0.62 | -1.76 ± 0.23 | -4.51 ± 0.90 | 0.13 ± 0.05 |
| NN-5 | - | 0.70 ±0.25 | 0.55 ±0.18 | -0.77 ±0.63 | -3.03 ± 0.58 | -7.00 ± 1.92 | 0.13 ± 0.05 |
| SVDKL-5 | -1.66 ± 0.26 | 0.40 ± 0.12 | 0.39 ± 0.12 | -1.63 ± 0.28 | -1.68 ± 0.25 | -2.30 ± 0.00 | 0.02 ± 0.01 |
| pNN-5 | - | 0.70 ± 0.25 | 0.55 ± 0.18 | -0.77 ± 0.63 | -3.22 ± 0.68 | -7.34 ± 2.06 | 0.13 ± 0.05 |
| fSVDKL-5 | -0.79 ± 0.62 | 0.70 ± 0.24 | 0.55 ± 0.18 | -0.77 ± 0.63 | -1.61 ± 0.29 | -4.25 ± 0.80 | 0.09 ± 0.04 |
| pSVDKL-5 | -0.77 ± 0.62 | 0.70 ± 0.24 | 0.55 ± 0.19 | -0.77 ± 0.63 | -1.64 ± 0.27 | -4.66 ± 0.96 | 0.11 ± 0.04 |
| NN-10 | - | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -2.05 ± 0.03 | -8.87 ± 0.10 | 0.18 ± 0.00 |
| SVDKL-10 | -0.76 ± 0.28 | 0.76 ± 0.09 | 0.63 ± 0.03 | -0.71 ± 0.28 | -1.37 ± 0.10 | -3.38 ± 0.77 | 0.10 ± 0.05 |
| pNN-10 | - | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -2.30 ± 0.11 | -9.48 ± 0.30 | 0.19 ± 0.00 |
| fSVDKL-10 | -0.02 ± 0.00 | 1.00 ± 0.00 | 0.78 ± 0.00 | -0.01 ± 0.00 | -1.14 ± 0.00 | -5.10 ± 0.01 | 0.14 ± 0.00 |
| pSVDKL-10 | -0.00 ± 0.00 | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -1.13 ± 0.01 | -5.24 ± 0.05 | 0.15 ± 0.00 |
| NN-20 | - | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -2.06 ± 0.02 | -8.91 ± 0.14 | 0.18 ± 0.00 |
| SVDKL-20 | -0.30 ± 0.20 | 0.91 ± 0.06 | 0.70 ± 0.02 | -0.26 ± 0.19 | -1.46 ± 0.16 | -4.72 ± 0.88 | 0.16 ± 0.04 |
| pNN-20 | - | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -2.24 ± 0.01 | -9.37 ± 0.08 | 0.19 ± 0.00 |
| fSVDKL-20 | -0.02 ± 0.00 | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.01 ± 0.00 | -1.14 ± 0.02 | -5.16 ± 0.10 | 0.13 ± 0.00 |
| pSVDKL-20 | -0.00 ± 0.00 | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -1.09 ± 0.01 | -5.12 ± 0.05 | 0.15 ± 0.00 |
| NN-50 | - | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -2.18 ± 0.01 | -9.23 ± 0.04 | 0.19 ± 0.00 |
| SVDKL-50 | -2.30 ± 0.00 | 0.10 ± 0.00 | 0.10 ± 0.00 | -2.30 ± 0.00 | -2.30 ± 0.00 | -2.30 ± 0.00 | 0.00 ± 0.00 |
| pNN-50 | - | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -2.38 ± 0.06 | -9.73 ± 0.15 | 0.19 ± 0.00 |
| fSVDKL-50 | -0.02 ± 0.00 | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -1.22 ± 0.01 | -5.48 ± 0.05 | 0.14 ± 0.00 |
| pSVDKL-50 | -0.00 ± 0.00 | 1.00 ± 0.00 | 0.79 ± 0.00 | -0.00 ± 0.00 | -1.11 ± 0.01 | -5.13 ± 0.04 | 0.15 ± 0.00 |

## D TABULATED UCI RESULTS

Here we tabulate the results for the UCI datasets.

Table 3: Results for BOSTON. We report means plus or minus one standard error averaged over the splits.

| | loss | train RMSE | test RMSE | train LL | test LL |
|---|---|---|---|---|---|
| SVGP | 1.66 ± 0.06 | 0.39 ± 0.01 | 0.37 ± 0.02 | -0.34 ± 0.01 | -0.33 ± 0.05 |
| fNN | 0.01 ± 0.00 | 0.02 ± 0.00 | 0.39 ± 0.03 | 2.28 ± 0.03 | -132.41 ± 22.39 |
| sNN | 0.01 ± 0.00 | 0.10 ± 0.00 | 0.34 ± 0.02 | 0.93 ± 0.02 | -5.61 ± 1.03 |
| DKL | -2.47 ± 0.00 | 0.00 ± 0.00 | 0.41 ± 0.02 | 2.72 ± 0.00 | -67.55 ± 3.97 |
| SVDKL | -0.47 ± 0.01 | 0.13 ± 0.00 | 0.35 ± 0.02 | 0.57 ± 0.01 | -1.12 ± 0.24 |

Table 4: Results for ENERGY.

| | loss | train RMSE | test RMSE | train LL | test LL |
|---|---|---|---|---|---|
| SVGP | 0.07 ± 0.01 | 0.19 ± 0.00 | 0.20 ± 0.00 | 0.19 ± 0.01 | 0.15 ± 0.02 |
| fNN | 0.00 ± 0.00 | 0.02 ± 0.00 | 0.04 ± 0.00 | 2.55 ± 0.02 | -0.04 ± 0.38 |
| sNN | 0.00 ± 0.00 | 0.02 ± 0.00 | 0.05 ± 0.00 | 2.31 ± 0.02 | 0.62 ± 0.19 |
| DKL | -3.01 ± 0.02 | 0.01 ± 0.00 | 0.05 ± 0.00 | 3.15 ± 0.02 | -2.63 ± 0.49 |
| SVDKL | -1.21 ± 0.00 | 0.03 ± 0.00 | 0.04 ± 0.00 | 1.26 ± 0.00 | 1.22 ± 0.01 |

Table 5: Results for KIN40K.

| | loss | train RMSE | test RMSE | train LL | test LL |
|---|---|---|---|---|---|
| SVGP | -0.14 ± 0.00 | 0.16 ± 0.00 | 0.17 ± 0.00 | 0.36 ± 0.00 | 0.33 ± 0.00 |
| fNN | 0.01 ± 0.00 | 0.03 ± 0.00 | 0.05 ± 0.00 | 2.18 ± 0.00 | 1.17 ± 0.02 |
| sNN | 0.01 ± 0.00 | 0.03 ± 0.00 | 0.05 ± 0.00 | 2.03 ± 0.00 | 1.51 ± 0.01 |
| VDKL | -1.41 ± 0.00 | 0.02 ± 0.00 | 0.05 ± 0.00 | 1.44 ± 0.00 | 1.33 ± 0.00 |
| SVDKL | -2.62 ± 0.00 | 0.01 ± 0.00 | 0.03 ± 0.00 | 2.68 ± 0.00 | 1.73 ± 0.02 |

Table 6: Results for POWER.

| | loss | train RMSE | test RMSE | train LL | test LL |
|---|---|---|---|---|---|
| SVGP | -0.01 ± 0.00 | 0.23 ± 0.00 | 0.23 ± 0.00 | 0.06 ± 0.00 | 0.07 ± 0.01 |
| fNN | 0.04 ± 0.00 | 0.17 ± 0.00 | 0.21 ± 0.00 | 0.37 ± 0.00 | 0.11 ± 0.02 |
| sNN | 0.05 ± 0.00 | 0.21 ± 0.00 | 0.22 ± 0.00 | 0.14 ± 0.00 | 0.11 ± 0.01 |
| VDKL | -0.57 ± 0.00 | 0.13 ± 0.00 | 0.21 ± 0.00 | 0.62 ± 0.00 | -0.02 ± 0.02 |
| SVDKL | -0.25 ± 0.00 | 0.18 ± 0.00 | 0.21 ± 0.00 | 0.28 ± 0.00 | 0.16 ± 0.01 |

Table 7: Results for PROTEIN.

| | loss | train RMSE | test RMSE | train LL | test LL |
|---|---|---|---|---|---|
| SVGP | 1.06 ± 0.00 | 0.64 ± 0.00 | 0.66 ± 0.00 | -0.98 ± 0.00 | -1.00 ± 0.00 |
| fNN | 0.19 ± 0.00 | 0.39 ± 0.00 | 0.58 ± 0.00 | -0.46 ± 0.00 | -1.09 ± 0.01 |
| sNN | 0.17 ± 0.00 | 0.35 ± 0.00 | 0.55 ± 0.00 | -0.36 ± 0.00 | -1.14 ± 0.01 |
| VDKL | 0.32 ± 0.01 | 0.30 ± 0.00 | 0.59 ± 0.00 | -0.23 ± 0.01 | -1.86 ± 0.01 |
| SVDKL | 0.35 ± 0.00 | 0.31 ± 0.00 | 0.57 ± 0.00 | -0.26 ± 0.00 | -1.29 ± 0.01 |

## References


Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Luis Pedro Coelho. Jug: Software for parallel reproducible computation in Python. *Journal of Open Research Software*, 5 (1), 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016.

James Hensman, Alexander G de G Matthews, Maurizio Filippone, and Zoubin Ghahramani. MCMC for variationally sparse gaussian processes. In *Advances in Neural Information Processing Systems*, pages 1648–1656, 2015.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Alexander G de G Matthews, , Mark van der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, 2017.

Christopher J Moore, Alvin JK Chua, Christopher PL Berry, and Jonathan R Gair. Fast methods for training Gaussian processes on large datasets. *Royal Society open science*, 3(5):160125, 2016.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264, 2006.

Michalis Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 567–574. PMLR, 2009.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *International Conference on Artificial Intelligence and Statistics*, pages 370–378. PMLR, 2016.

Zhifei Zhang, Yang Song, and Hairong Qi. Age progression/regression by conditional adversarial autoencoder. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.